THE UNIVERSITY OF
MELBOURNE

COMP 90048 Declarative Programming Workshop 2 (week3)

2019 semester 1

by Wendy Zeng

Tutorial : Tue 18:15 - 19:15 221 Bouverie St, room B11
Wed 17:15 - 18:15 201 Bouverie St, room B132

Outline

1. Pattern matching and quizzes (1&2) recap
2. Type constructor and data constructor (Q2)
3. Recursion VS Loop (Q6)
4. Coding practice time (Q3, Q4, Q5, Q7)

1. Pattern matching and quizzes (1&2) recap

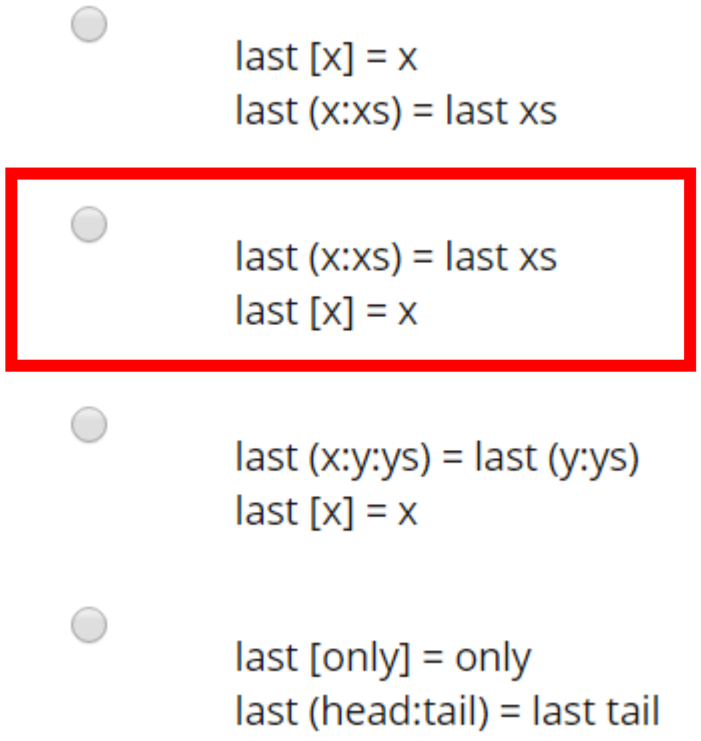
- Pattern matching principles:
  - **Exhaustive**: at least one pattern should

apply for any possible call

  - **Exclusive:** at most one pattern should apply for any possible call

- Evaluation order:
  - Top-down, left to right

1. Pattern matching and quizzes (1&2) recap



○
```
last [x] = x
last (x:xs) = last xs
```

Correct
Earlier pattern does not overlap with later one

○
```
last (x:xs) = last xs
last [x] = x
```

Wrong
Earlier pattern overlaps with later one

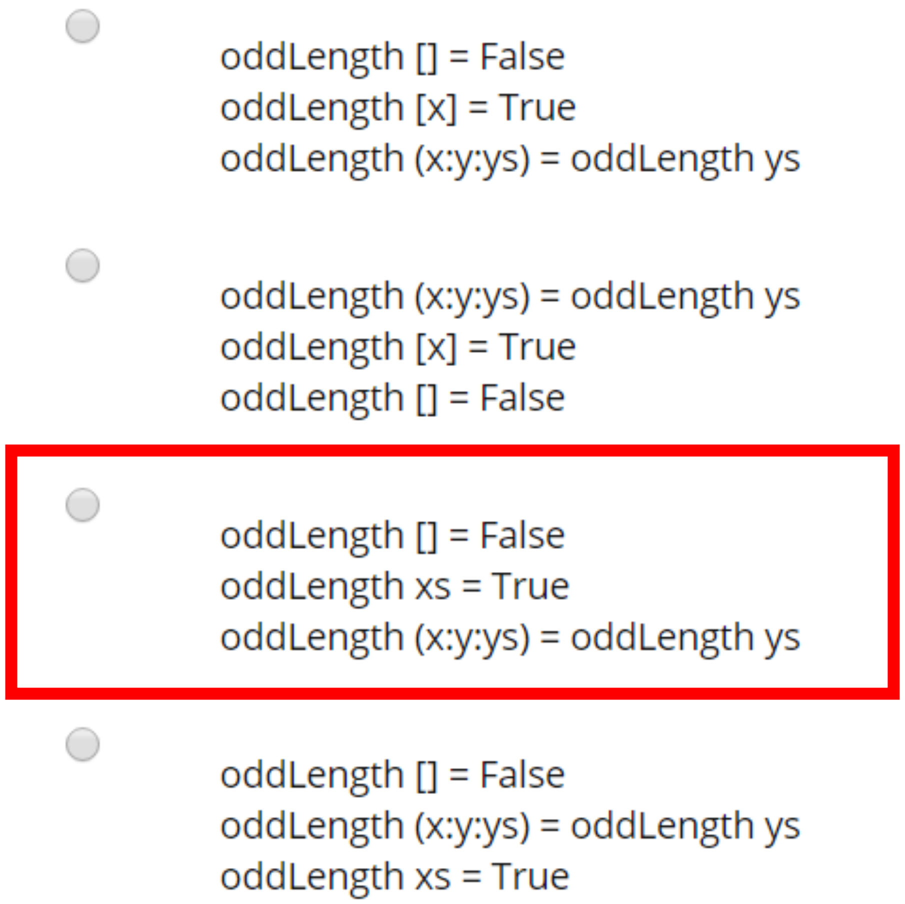
○
```
last (x:y:ys) = last (y:ys)
last [x] = x
```

Correct
Exclusive patterns in each equation

○
```
last [only] = only
last (head:tail) = last tail
```

Correct
Same as option A

1. Pattern matching and quizzes (1&2) recap



○
```
oddLength [] = False
oddLength [x] = True
oddLength (x:y:ys) = oddLength ys
```

Correct
Exhaustive and exclusive

○
```
oddLength (x:y:ys) = oddLength ys
oddLength [x] = True
oddLength [] = False
```

Correct
Exhaustive and exclusive

○
```
oddLength [] = False
oddLength xs = True
oddLength (x:y:ys) = oddLength ys
```

Wrong
Second pattern xs is inclusive of (x:y:xs) and will shadow the recursive equation

○
```
oddLength [] = False
oddLength (x:y:ys) = oddLength ys
oddLength xs = True
```

Correct
Last pattern xs is inclusive of previous two, but it will be matched latest and will not shadow the previous two equation

2. Type Constructor & Data Constructor

```
data TypeCons TypeArg1 TypeArg2 … =  DataCons1 TypeArg1 TypeArg2 ..
                                        | DataCons2 TypeArg3 TypeArg4
…
                                        | DataCons3 TypeArg5 TypeArg6
…
                                         …
```

Type Constructor: constructs type

Data Constructor: constructs values of certain type

2. Type Constructor & Data Constructor

```
data Suit = Club | Diamond | Heart | Spade
```

- Data constructor with no arguments can be seen as constants

```
data Card = Card Suit Rank
```

- Data constructors are **algebraic data type** (AND relationship: **grouping of different values**, s.t. Suit and Rank satisfy AND relationship; OR relationship: alternate between data constructors, s.t. **either Club or Diamond can’t be both**)

```
data JCard = NormalCard Suit Rank | JokerCard JokerColor
```

- Discriminate union type

2. Type Constructor & Data Constructor

- Both type constructors and data constructors can have zero ore more of other types as **arguments**

data Tree a = Leaf | Node a (Tree a) (Tree a)	**Binary tree**
data List a = Empty | Node a (List a)	**Linked list**
data Mtree a = Mnode a [Mtree a]	**Meta tree**

3. Recursion VS Loop

```
loopCtrl :: (Int, Int) -> (Int, Int)
loopCtrl (n, 0) = (n, 0)
loopCtrl (n, c)
          | n > 100 = loopCtrl (n-10, c-1)
          | otherwise = loopCtrl (n+11, c+1)

mccarthy91 :: Int -> Int
mccarthy91 n = newN
 where (newN, _) = loopCtrl (n, 1)
```

THE UNIVERSITY OF
MELBOURNE

Thank you

wendy.zeng@unimelb.edu.au

By Wendy Zeng